Positive higher-order queries

Michael Benedikt, Gabriele Puppis, Huy Vu

▶ **To cite this version:**

Michael Benedikt, Gabriele Puppis, Huy Vu. Positive higher-order queries. Proceedings of PODS 2010, 2010, Indianapolis, United States. pp.27-38, 10.1145/1807085.1807091 . hal-00717792

HAL Id: hal-00717792
https://hal.science/hal-00717792

Submitted on 30 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Positive Higher-Order Queries

Michael Benedikt
Oxford University
Computing Laboratory
Parks Road, Oxford, UK
michael.benedikt@comlab.ox.ac.uk

Gabriele Puppis
Oxford University
Computing Laboratory
Parks Road, Oxford, UK
gabriele.puppis@comlab.ox.ac.uk

Huy Vu
Oxford University
Computing Laboratory
Parks Road, Oxford, UK
huy.vu@comlab.ox.ac.uk

ABSTRACT

We investigate a higher-order query language that embeds operators of the positive relational algebra within the simply-typed λ-calculus. Our language allows one to succinctly define ordinary positive relational algebra queries (conjunctive queries and unions of conjunctive queries) and, in addition, *second-order query functionals*, which allow the transformation of CQs and UCQs in a generic (i.e., syntax-independent) way. We investigate the equivalence and containment problems for this calculus, which subsumes traditional CQ/UCQ containment. Query functionals are said to be equivalent if the output queries are equivalent, for each possible input query, and similarly for containment. These notions of containment and equivalence depend on the class of (ordinary relational algebra) queries considered. We show that containment and equivalence are decidable when query variables are restricted to positive relational algebra and we identify the precise complexity of the problem. We also identify classes of functionals where containment is tractable. Finally, we provide upper bounds to the complexity of the containment problem when functionals act over other classes.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Logical Design, Languages—*data models, query languages*; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

General Terms

Theory

Keywords

complexity, algorithms

1. INTRODUCTION

Query transformation is a basic operation in database systems. In processing queries over views, query rewriting is a fundamental tool – queries over the view are rewritten to queries over base data. In query relaxation [21, 3] queries are rewritten to get a larger class of results. Another topic of recent interest is *query specification* [24, 35, 11], which can be seen as boolean querying of queries. Query specification is an approach to specify permitted queries to secure access to web datasources.

The importance of query transformation makes it natural for us to consider a query language for querying queries. In this work, we will examine *higher-order query language*, based on terms that feature both variables ranging over queries and variables ranging over relations. Terms are built up via the normal relational algebra operations, plus a new operation application of a query variable to an expression. Higher-order terms can be considered in two ways: as functions of the first-order and second-order variables together, or (via currying the second-order variables) as mappings from queries to queries.

EXAMPLE 1. *We consider transformations that transform input queries P, Q, where both P and Q take as input relations R with integer-valued attributes a and b and return relations with the same schema. One such transformation takes P and Q and returns the query $\sigma_{a=5}(P \cap Q)$. This would be expressible in our language as $\lambda P.\, \lambda Q.\, \lambda R.\ \sigma_{a=5}\big(P(R) \bowtie Q(R)\big)$. Another such transformation takes P and Q and returns the query $\sigma_{a=5} \circ Q \circ P$. This would be expressible in our language as $\lambda P.\, \lambda Q.\, \lambda R.\ \sigma_{a=5}\big(Q(P(R))\big)$.*

In the above examples, P and Q are *query variables* while R is a first-order variable – R ranges over finite databases for a schema with attributes $\{a, b\}$, while P and Q range over mappings between such databases.

We look for languages with two important properties. The first is that the transformations defined in our languages, as in the examples above, are *generic* – the output of a term when the higher-order variables are bound to queries depends only on the semantics of the queries. This is in contrast to query transformation and specification languages which allow direct access to the syntax of the queries [26, 35]. Secondly, we search for languages where *static analysis and optimization are possible*, extending techniques from the case of standard selection project join queries in the relational case. This is again in contrast to prior languages for querying queries (e.g. [26]), which are relationally complete, and hence cannot admit static guarantees even of satisfiability.

This second goal impacts our calculus in two ways: it influences what mappings the query variables P and Q range

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODS'10, June 6–11, 2010, Indianapolis, Indiana, USA.
Copyright 2010 ACM 978-1-4503-0033-9/10/06 ...$10.00.

over, and also what relational algebra operators are permitted in addition to application. We will look at higher-order languages where queries range over a tame fragment of relational algebra. We thus focus on queries in higher-order languages that generalize *positive relational algebra*, rather than full relational algebra. The restriction to positive queries will hold both for constants used to build terms and for query variables. We will define several variants of positive higher-order query languages and investigate the containment problem for them. We will show that many important containment and equivalence problems are decidable in the case of queries ranging over positive relational algebra. We will also look at the containment problem for the ordinary data-to-data queries built up in this language.

Our contributions can be summarized as follows:

1. We define a higher-order query language for which several basic analysis problems are decidable (Section 2), along with a particularly simple expressively equivalent subset of the language – the normal-form queries.
2. We isolate the complexity of the containment and equivalence problems for higher-order queries in normal-form that manipulate positive relational algebra queries, and also give results in the presence of dependencies (Section 3).
3. We give upper bounds to the complexity of the containment and equivalence problems for normal-form higher-order queries over other bases (Section 4).
4. We give preliminary results related to terms (higher-order and lower-order) that are not in normal form (Section 5).

Due to space limitations, proofs are either sketched or deferred for the full version.

2. DEFINITIONS

2.1 Types

We fix an infinite set of *attribute names* (or *attributes*). We define the *relational types* as the (possibly empty) tuples of attribute names, $\mathcal{T} = (a_1, ..., a_m)$, for any $m \in \mathbb{N}$ (the type corresponding to the empty tuple is denoted ε). We manipulate relational types by using the standard operations on tuples, such as the juxtaposition (without duplicates) $\mathcal{T} + \mathcal{T}'$ and the projection $\pi_A(\mathcal{T})$, for a given set A of attributes in $\mathcal{T}$.

Relational types are the basic building blocks of more complex types. We define higher-order types, hereafter called *query types*, by using the functional type constructor: if $\mathcal{T}, \mathcal{T}'$ are (relational or query) types, then $\mathcal{T} \to \mathcal{T}'$ is a query type. As usual, we assume that the functional type constructor is right-associative and we view any query type of the form $\mathcal{T}_1 \to ... \to \mathcal{T}_m \to \mathcal{T}'$ as the curried form of the functional type $(\mathcal{T}_1 \times ... \times \mathcal{T}_m) \to \mathcal{T}'$,

We define the *order* of a type $\mathcal{T}$, denoted $\mathtt{order}(\mathcal{T})$, as follows: we let $\mathtt{order}(\mathcal{R}) = 0$ for any relational type $\mathcal{R}$ and $\mathtt{order}(\mathcal{T} \to \mathcal{T}') = \max\big(\mathtt{order}(\mathcal{T}) + 1, \mathtt{order}(\mathcal{T}')\big)$ for any query type $\mathcal{T} \to \mathcal{T}'$.

We associate with each attribute name a_i a range $\mathcal{D}om(a_i)$ of possible values, called the *attribute range* of a_i. Examples of attribute ranges are the integers $\mathbb{Z}$ and the booleans $\mathbb{B}$. We assume that there are infinitely many attribute names associated with each attribute range. The elements in each attribute range $\mathcal{D}om(a_i)$ are called *attribute values*. Similarly, given a relational type $\mathcal{R} = (a_1, ..., a_m)$, we denote by $\mathcal{D}om(\mathcal{R})$ the set $\mathcal{D}om(a_1) \times ... \times \mathcal{D}om(a_m)$, whose elements are called *records*. Given a record $t \in \mathcal{D}om(a_1) \times ... \times \mathcal{D}om(a_m)$, we denote by $t.a_i$ the value of the attribute a_i in t.

The instances of a relational type $\mathcal{R}$, which are called *relations*, are the finite sets $\mathtt{R}$ consisting of some records chosen from $\mathcal{D}om(\mathcal{R})$ (note that there are only two relations of type ε, namely, the empty set, usually identified with the boolean value `false`, and the singleton $\{\varepsilon\}$, usually identified with the boolean value `true`). In a similar way, the instances of a query type $\mathcal{T} \to \mathcal{T}'$, called *queries*, are the functions $\mathtt{Q}$ that maps objects x of type $\mathcal{T}$ to objects of type $\mathcal{T}'$. We will be mainly concerned with types of order at most 2 in this work. An an example, queries of order 1 map tuples of relations to relations, while queries of order 2 map tuples of queries to queries.

2.2 Terms and their semantics

We now define our variant of the simply-typed λ-calculus for the setting where we can abstract either over relations or queries.

First of all, we fix a *signature* $\mathcal{F}$, namely, a set of relational constants and query constants together with the associated arities. We use RA^+ to denote the signature for Positive Relational Algebra, which contains the following constants: (i) all finite relations $\mathtt{R}$, viewed as constants of order 0 – we often abuse notation by identifying each constant symbol with its interpretation; (ii) the unary rename operator $\rho_{a/b}$, which renames the attribute a by b in a given input relation; (iii) the unary operator π_A, which projects an input relation into the subset A of its attributes; (iv) the unary operator σ_c, which selects a subset of the tuples from a given relation according to the condition c envisaging equalities between attributes/constants; (v) the binary operator $\bowtie$, which returns the cartesian product of two input relations followed by a selection of the tuples that have the same values on the same attribute names; (vi) the binary operator $\cup$, which returns the union of two input relations of the same type. Another signature of particular interest is that of Conjunctive Queries, denoted CQ, which consists of the four families of operators $\rho_{a/b}$, π_A, σ_c, and $\bowtie$ of the Relational Algebra, and of Conjunctive Queries with Relational Constants $\mathsf{CQ_C}$, which adds to CQ all relational instances as constants. Finally RA extends RA^+ with the usual difference operator $\setminus$. We sometimes use the infix notation for the constants of arity 2 (for instance, for the operators $\bowtie$ and $\cup$).

We also fix an infinite set $\mathcal{X}$ of relational and query variables. Sometimes, we may omit the type of a variable when it is clear from the context (for instance, we will usually denote relational variables by $R, R', ...$ and order 1 query variables by $Q, Q', ...$).

Higher-order *terms* are build up from constants in $\mathcal{F}$ and variables in $\mathcal{X}$ by using the operations of abstraction and application: if X is a variable of type $\mathcal{T}$ and φ is a term of type $\mathcal{T}'$, then $\lambda X.\varphi$ is a term of type $\mathcal{T} \to \mathcal{T}'$; similarly, if Φ is a term of type $\mathcal{T} \to \mathcal{T}'$ and φ is a term of type $\mathcal{T}$, then $\Phi(\varphi)$ is a term of type $\mathcal{T}'$. We say that a term Φ is *closed* if it contains no free occurrences of variables. The operation $\Phi_1 \circ \Phi_2$ of functional composition is often used as a shorthand for $\lambda X.\, \Phi_1(\Phi_2(X))$, provided that the resulting term is well-typed.

Given a term Φ, we define the *order* of Φ as the order of its type and the *degree* of Φ as the maximum order of its subterms. As an example, $\big(\lambda Q.\,\lambda R.\,Q(R)\big)(\pi_A)$ is a term of order 1 and degree 2. We also define the *size* of a term inductively as follows. The size of a relational constant is the size of the corresponding instance, namely, the number of attributes times number of rows. The size of a query constant is its length. The size of a first-order or a second-order variable is 1. The size of a higher-order term is defined as 1 plus the sum of the sizes of its top-level sub-terms.

As for the semantics of terms, the obvious evaluation method is to pair the standard operational semantics of the λ-calculus with an interpretation for the relational constants and the query constants. Below, we define such a semantics by exploiting an induction on the order of terms.

In order to do that, we need to first fix an interpretation for the constants and the variable domains. Formally, an interpretation $\mathcal{I}$ for the signature $\mathcal{F}$ is a function that maps (i) every constant $\mathtt{const} \in \mathcal{F}$ to its semantics $[\![\mathtt{const}]\!]_{\mathcal{I}}$ (e.g., $[\![\cup]\!]_{\mathcal{I}}$ is usually the function that maps a pair of relations $\mathtt{R}_1$ and $\mathtt{R}_2$ to their union $\mathtt{R}_1 \cup \mathtt{R}_2$) and (ii) every variable $X \in \mathcal{X}$ to its domain $\mathcal{D}om_{\mathcal{I}}(X)$ (e.g., if X is an order 1 query variable, then $\mathcal{D}om_{\mathcal{I}}(X)$ can be the set of all queries of the Positive Relational Algebra). Below, we make the underlying interpretation $\mathcal{I}$ explicit by denoting the semantics of a term Φ by $[\![\Phi]\!]_{\mathcal{I}}$.

For every term Φ of the form $\mathtt{const}(\varphi_1, ..., \varphi_k)$, where $k \in \mathbb{N}$ is the arity of the constant $\mathtt{const} \in \mathcal{F}$, we denote by $[\![\Phi]\!]_{\mathcal{I}}$ the relation $[\![\mathtt{const}]\!]_{\mathcal{I}}\big([\![\varphi_1]\!]_{\mathcal{I}}, ..., [\![\varphi_1]\!]_{\mathcal{I}}\big)$. Similarly, given a term Φ of the form $\lambda X.\,\varphi(X)$, we denote by $[\![\Phi]\!]_{\mathcal{I}}$ the function that maps every object x in $\mathcal{D}om_{\mathcal{I}}(X)$ to the object $[\![\varphi]\!]_{\mathcal{I}[X/x]}$, where $\mathcal{I}[X/x]$ is the interpretation for the extended signature $\mathcal{F} \cup \{x\}$ obtained from $\mathcal{I}$ by letting $[\![x]\!]_{\mathcal{I}[X/x]} = x$ be the interpretation for the new constant x. Finally, given a term Φ of the form $\varphi_1(\varphi_2)$, we denote by $[\![\Phi]\!]_{\mathcal{I}}$ the object $[\![\varphi_1]\!]_{\mathcal{I}}\big([\![\varphi_2]\!]_{\mathcal{I}}\big)$.

From now on, for a fixed signature $\mathcal{F}$ (e.g., $\mathcal{F} = \mathsf{RA}^+$), we tacitly assume the standard interpretation for the constants in $\mathcal{F}$ and the standard interpretation for the domains of the relational variables, which are the sets of finite relations of appropriate types. We now explain how the ordinary relational calculus embeds in our language. A term is *simple* if it contains no second-order variables and no λ-abstractions: thus, a simple term is formed by just using the constants of the signature. We identify a simple term with the query obtained by abstracting all of its relational variables and adding a fresh abstracted variable if there are none free. Under this convention RA terms correspond to Relational Algebra queries in the usual sense, RA^+ terms correspond to Positive Relational Algebra queries, and CQ terms correspond to select-project-join queries [1]. The signature CQ_C extends CQ with the set of all relational constants. We will freely use RA, RA^+, CQ_C, and CQ to refer to both the simple terms and the associated queries.

In contrast to the case of relation variables, we let the domains for *query variables* be unspecified a priori, and we use an auxiliary argument to completely describe their semantics. We shall denote by $\lambda\mathsf{RA}^+$ (resp., $\lambda\mathsf{CQ}_\mathsf{C}$, $\lambda\mathsf{RA}$) the interpretation for $\mathcal{F}$ that associates with any order 1 variable Q the set of all queries of the Positive Relational Algebra (resp., the set of all Conjunctive Queries with Relational Constants, the set of all Relational Algebra queries). We will sometimes refer to the range of variables as the *base*. As an example, if $\Phi = \lambda Q.\,\lambda R.\,Q(R)$, then $[\![\Phi]\!]_{\lambda\mathsf{RA}^+}$ denotes the function that maps a query $\mathtt{Q}$ of the Positive Relational Algebra and a finite relation $\mathtt{R}$ to the finite relation $\mathtt{Q}(\mathtt{R})$. Moreover, if the interpretation $\mathcal{I}$ is clear from the context, we can omit the subscript $\mathcal{I}$ from $[\![\Phi]\!]_{\mathcal{I}}$. By a slight abuse of notation, we can also write $\mathtt{const}$ in place of $[\![\mathtt{const}]\!]$ for the standard interpretation of the constant $\mathtt{const}$ in the signature $\mathcal{F}$.

2.3 Normal forms

We recall the notions of β-reduction, η-expansion, and η-long β-normal form. We identify terms up to α-congruence, that is, we identify any two terms of the form $\lambda X.\,\varphi$ and $\lambda Y.\,\varphi[X/Y]$, where $\varphi[X/Y]$ denotes the substitution of every free occurrence of the variable X in φ by a fresh variable Y. We call *β-reduction* the application, in any given context, of the following rewriting rule (renaming of bound variables may be necessary in order to avoid variable capture):

$$(\lambda X.\Phi)(\varphi) \quad \rightsquigarrow \quad \Phi[X/\varphi].$$

The lefthandside term above is called a *redex*. A term is said to be in *β-normal form* if it contains no redex (and hence no β-reduction can be applied to it).

Another useful transformation is that of *η-expansion*, which transforms a subterm Φ of functional type $\mathcal{T} \to \mathcal{T}'$ to the subterm $\lambda X.\,\Phi(X)$, where X is a fresh variable of type $\mathcal{T}$. In order to guarantee termination, the operation of η-expansion is restricted to the subterms Φ that do not start with the abstraction operator λ and that have no explicit argument in their context (e.g., η-expansion is never applied to the subterms Φ when they occur in a context like $\Phi(\varphi)$). A term is said to be in *η-long β-normal form* (hereafter, simply *normal form*) if no β-reduction nor η-expansion (as restricted before) is possible.

Since the operations of β-reduction and η-expansion are confluent and always terminating (on well-typed terms), we have that every term Φ has a *unique* normal form, denoted $\Phi^{\downarrow}$. Moreover, the normal form of a term can be obtained by first applying all β-reductions and then all η-expansions. This also shows that the normal form of any term Φ of order 2 can be written as follows:

$$\Phi^{\downarrow} \;=\; \lambda Q_1 ... \lambda Q_m.\; \lambda R_1 ... \lambda R_n.\; \varphi$$

where $Q_1, ..., Q_m$ are order 1 query variables, $R_1, ..., R_n$ are relational variables, and φ is a term of order 0 with free variables among $Q_1, ..., Q_m$, $R_1, ..., R_n$, but with no occurrence of λ-abstraction. In particular if Φ is a closed term of relational type, then the normal form is just a term of relational type built up from constants, which can then be evaluated, using the semantics of the constants to get a relation. Thus we have a (naïve but) effective way of evaluating closed terms.

2.4 The term hierarchy

We introduce some notation that will be extensively used through the rest of the paper.

DEFINITION 1. *Let $\mathcal{F}$ be a generic signature and let m, n be two natural numbers such that $m \leq n$. We denote by*

- $\mathsf{Terms}_{m,n}[\mathcal{F}]$ *the class of all closed terms of order m and degree n that are built up from constants in the signature $\mathcal{F}$ using abstraction and application,*

- $\mathsf{Terms}^{\downarrow}_{m}[\mathcal{F}]$ *the subclass of* $\mathsf{Terms}_{m,n}[\mathcal{F}]$ *consisting only of terms in normal form (note that the degree and the order coincide for terms in normal form).*

As an example, $\mathsf{Terms}_{0,1}[\mathsf{RA}^+]$ (resp., $\mathsf{Terms}_{0,1}[\mathsf{CQ}]$) is the class of all closed terms of relational type (e.g., $\Phi = (\lambda R.\, R \bowtie \rho_{a/b}(R))\{t_0, t_1\}$) that are built up from the operators of the Positive Relational Algebra (resp., from the operators $\rho_{a/b}$, π_A, σ_c, and $\bowtie$) via application and abstraction over variables of degree at most 1. Note that normal forms of terms of order 1 are the same as simple terms; hence the class $\mathsf{Terms}^{\downarrow}_{1}[\mathsf{RA}^+]$ coincides exactly with what we have called RA^+ above, and similarly for RA, RA^+, CQ_{C} – we will thus use these notations interchangeably. We will also use UCQ to denote the simple terms (or, equivalently, order 1 terms in normal form) that are built up from the signature $\mathsf{RA}^+$by only using singleton relational constants and by allowing the union operator to appear only at the topmost level. Such a class translates efficiently to Unions of Conjunctive Queries.

2.5 The containment problem

We now come to the main topic of this paper: we introduce a generalization of the *containment* relation $\subseteq$ between terms and we define the main static analysis problem we will deal with in the paper. From now on, $\mathcal{C}$ and $\mathcal{C}'$ will denote two generic classes of terms and $\mathcal{I}$ an interpretation for them.

For terms of order 0, the definition of containment is straightforward: given two closed terms Φ and Φ' of the same relational type, we write $\Phi \subseteq_{\mathcal{I}} \Phi'$ iff $[\![\Phi]\!]_{\mathcal{I}} \subseteq [\![\Phi']\!]_{\mathcal{I}}$ (note that the underlying interpretation $\mathcal{I}$ for fragments of the Relational Algebra will be often omitted).

We then extend the definition of containment from relational terms to order $n > 0$ queries as follows. Given two closed terms $\Phi = \lambda X.\, \varphi$ and $\Phi' = \lambda X.\, \varphi'$ of the same query type $\mathcal{T} \to \mathcal{T}'$, we write

$$\Phi \subseteq_{\mathcal{I}} \Phi' \quad \text{iff} \quad \forall\, x \in \mathcal{D}om_{\mathcal{I}}(X)\ .\ \Phi(x) \subseteq_{\mathcal{I}} \Phi'(x).$$

As an example, given two order 2 terms Φ and Φ' of the same type, we write $\Phi \subseteq_{\lambda\mathsf{RA}^+} \Phi'$ iff, for all instances $\mathtt{Q}_1, ..., \mathtt{Q}_m$, $\mathtt{R}_1, ..., \mathtt{R}_n$ of the formal arguments $Q_1, ..., Q_m$, $R_1, ..., R_n$ in Φ and Φ', with each $\mathtt{Q}_i$ ranging over the set of queries of Positive Relational Algebra and each $\mathtt{R}_i$ ranging over the set of finite relations, we have $[\![\Phi]\!](\mathtt{Q}_1, ..., \mathtt{Q}_m, \mathtt{R}_1, ..., \mathtt{R}_n) \subseteq [\![\Phi']\!](\mathtt{Q}_1, ..., \mathtt{Q}_m, \mathtt{R}_1, ..., \mathtt{R}_n)$.

Definition 2. *The* containment problem *for lefthand-side terms in $\mathcal{C}$ and righthandside terms in $\mathcal{C}'$, under the interpretation $\mathcal{I}$, consists of deciding, given two terms $\Phi \in \mathcal{C}$ and $\Phi' \in \mathcal{C}'$ of the same type, whether $\Phi \subseteq_{\mathcal{I}} \Phi'$.*

It is worth remarking that the containment problem subsumes several crucial problems related to (higher-order) queries and, more generally, functional programs, such as *satisfiability* (i.e., given a term Φ, decide whether there is an input x such that $\Phi(x)$ evaluates to `true`) and the *extensional equivalence* (i.e., given Φ and Φ', decide whether $\Phi(x) = \Phi'(x)$ for every input x). As an example, two terms Φ and Φ' are extensionally equivalent, under an underlying interpretation $\mathcal{I}$, iff $\Phi \subseteq_{\mathcal{I}} \Phi'$ and $\Phi' \subseteq_{\mathcal{I}} \Phi$.

We will *always* consider the computational complexity of our problems in terms of the *size* of the terms, as defined earlier in this section.

We conclude the section with some examples that show how the containment relation may depend on the underlying interpretation for the domains of the query variables.

Example 2. *Let R be a variable of relational type $\mathcal{R} = (a)$, with $\mathcal{D}om(a) = \mathbb{Z}$, and let Q be a variable of query type $\mathcal{R} \to \mathcal{R}$. Consider the order 2 terms:*

$$\begin{aligned} \Phi &= \lambda Q.\, \lambda R.\, Q\big(Q(\sigma_{a=1}(R))\big) \\ \Phi' &= \lambda Q.\, \lambda R.\, Q\big(\sigma_{a=1}(R)\big) \end{aligned}$$

over the signature CQ. Take an arbitrary query constant $\mathtt{Q}$ and an arbitrary relational constant $\mathtt{R}$ as instances of Q and R. Note that $\sigma_{a=1}(\mathtt{R})$ is either a singleton or the empty set. If a CQ $\mathtt{Q}$ returns a non-empty relation on input $\sigma_{a=1}(\mathtt{R})$, then it must return a singleton consisting either of the tuple t_1, with $t_1.a = 1$, or the tuple t_2, with $t_2.a = c$, for some constant c that appears in $\mathtt{Q}$. Now, if $\mathtt{Q}\big(\sigma_{a=1}(\mathtt{R})\big) = \{t_1\}$, then, by monotonicity, we have $\mathtt{Q}\big(\mathtt{Q}\big(\sigma_{a=1}(\mathtt{R})\big)\big) = \{t_1\}$. Otherwise, if $\mathtt{Q}\big(\sigma_{a=1}(\mathtt{R})\big) = \{t_2\}$, then case analysis on $\mathtt{Q}$ shows that $\mathtt{Q}\big(\mathtt{Q}\big(\sigma_{a=1}(\mathtt{R})\big)\big)$ must be either the singleton $\{t_2\}$ or the empty set. Therefore, we have that Φ is contained in Φ' under the interpretation of the query variables by Conjunctive Queries, shortly, $\Phi \subseteq_{\lambda\mathsf{CQ}} \Phi'$. On the other hand, we have $\Phi \not\subseteq_{\lambda\mathsf{RA}^+} \Phi'$, since we can take $\mathtt{Q}$ such that $\mathtt{Q}\big(\{t_1\}\big) = \{t_2\}$ and $\mathtt{Q}\big(\{t_2\}\big) = \{t_3\}$, with $t_1.a = 1$, $t_2.a = 2$, and $t_3.a = 3$.

Example 3. *Let R be a variable of relational type $\mathcal{R} = (a)$, with $\mathcal{D}om(a) = \mathbb{Z}$, and let Q be a variable of query type $\mathcal{R} \to \varepsilon$. Consider the order 2 terms:*

$$\begin{aligned} \Phi &= \lambda Q.\, \lambda R.\, \pi_{\emptyset}\big(\sigma_{b=2}\big(Q(\sigma_{a=1}(R))\big)\big) \bowtie \\ &\qquad \pi_{\emptyset}\big(\sigma_{b=3}\big(Q(\sigma_{a=1}(R))\big)\big) \\ \Phi' &= \lambda Q.\, \lambda R.\, \pi_{\emptyset}\ \big(\sigma_{a=1}\big(\sigma_{a=2}(R)\big)\big) \\ &\qquad (\Phi' \text{ returns always } \mathtt{false}) \end{aligned}$$

over the signature CQ. When we instantiate Q by a CQ $\mathtt{Q}$, $\Phi(\mathtt{Q})$ turns out to be unsatisfiable, since for any instance $\mathtt{R}$ of R, we have $\sigma_{a=1}(\mathtt{R})$ is either a singleton or the empty set and hence $\sigma_{b=2}\big(\mathtt{Q}(\sigma_{a=1}(\mathtt{R}))\big)$ and $\sigma_{b=3}\big(\mathtt{Q}(\sigma_{a=1}(\mathtt{R}))\big)$ cannot return a non-empty set at the same time. However, if we choose $\mathtt{R} = \{t_1\}$ and $\mathtt{Q}$ to be a union of conjunctive queries in such a way that $\mathtt{Q}(\mathtt{R}) = \{t_2\} \cup \{t_3\}$, where $t_1.a = 1$, $t_2.a = 2$, and $t_3.a = 3$, then $\Phi(\mathtt{Q}, \mathtt{R})$ evaluates to `true`. This shows that $\Phi \subseteq_{\lambda\mathsf{CQ}} \Phi'$ and $\Phi \not\subseteq_{\lambda\mathsf{RA}^+} \Phi'$.

Example 4. *Let R_1, R_2 be two variables of relational type $\mathcal{R} = (a)$, with $\mathcal{D}om(a) = \mathbb{Z}$, and let Q be a variable of query type $\mathcal{R} \to \varepsilon$. Consider the order 2 terms:*

$$\begin{aligned} \Phi &= \lambda Q.\, \lambda R_1.\, \lambda R_2.\, Q(R_1) \\ \Phi' &= \lambda Q.\, \lambda R_1.\, \lambda R_2.\, Q(R_1 \cup R_2) \end{aligned}$$

over the signature RA^+. For every monotone query $\mathtt{Q}$ (and, in particular, for every query of the Positive Relational Algebra) and for every pair of relations $\mathtt{R}_1, \mathtt{R}_2$, we have $\mathtt{Q}(\mathtt{R}_1) \subseteq \mathtt{Q}(\mathtt{R}_1 \cup \mathtt{R}_2)$. Thus, $\Phi \subseteq_{\lambda\mathsf{RA}^+} \Phi'$. On the other hand, for any signature $\mathcal{F}$ that extends RA^+ with the difference operator $\setminus$, we have $\Phi \not\subseteq_{\lambda\mathcal{F}} \Phi'$, since we can choose $\mathtt{R}_1 = \{t_1\}$, $\mathtt{R}_2 = \{t_2\}$, with $t_1.a = 1$ and $t_2.a = 2$ as instances of R_1, R_2, and $\mathtt{Q} = \lambda S.\, \mathtt{true} \setminus \pi_{\emptyset}\big(\sigma_{a=2}(S)\big)$ as an instance of Q.

3. CONTAINMENT OF HIGHER-ORDER QUERIES: POSITIVE RELATIONAL ALGEBRA

The goal of this section will be to prove tight bounds on the complexity of the containment problem for order 2 terms

in normal form, namely, for higher-order queries, where the formal arguments (i.e., the query variables and the relational variables) are interpreted by terms of the Positive Relational Algebra.

3.1 The complexity of higher-order containment

The goal of this subsection is to prove:

THEOREM 1. *The problem of deciding the containment* $\Phi \subseteq_{\lambda \mathsf{RA}^+} \Phi'$*, where* $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{RA}^+]$*, is* Π_2^{P}*-complete.*

We will need to build up a bit of infrastructure first. We start by introducing some variants of the classical problem of deciding containment of CQs in UCQs. The main variation is that containment is relative to a set of constraints of the form $\mathtt{R}_i \subseteq \mathtt{R}_j$ (positive constraints) or $\mathtt{R}_i \not\subseteq \mathtt{R}_j$ (negative constraints), where $\mathtt{R}_i$ and $\mathtt{R}_j$ are relational symbols. Moreover, we introduce a disjunctive variant of the constrained containment problem.

DEFINITION 3.

- Constrained Containment Problem: *given two queries* $\mathtt{Q}, \mathtt{Q}'$ *of the same type* $\bar{\mathcal{R}} \to \mathcal{S}$ *and given a set* Σ *of constraints over appropriate relations for* $\bar{\mathcal{R}}$*, the problem consists of deciding whether* $[\![\mathtt{Q}]\!](\bar{\mathtt{R}}) \subseteq [\![\mathtt{Q}']\!](\bar{\mathtt{R}})$ *holds for all instances* $\bar{\mathtt{R}}$ *satisfying the constraints in* Σ*;*
- Constrained Disjunctive Containment Problem: *given some queries* $\mathtt{Q}_1, ..., \mathtt{Q}_n$ *and* $\mathtt{Q}'_1, ..., \mathtt{Q}'_n$*, having types* $\bar{\mathcal{R}} \to \mathcal{S}_1$*, ...,* $\bar{\mathcal{R}} \to \mathcal{S}_n$*, and given a set* Σ *of constraints over appropriate relations for* $\bar{\mathcal{R}}$*, the problem consists of deciding whether, for every instance* $\bar{\mathtt{R}}$ *satisfying* Σ*, there is an index* $1 \leq i \leq n$ *such that* $[\![\mathtt{Q}]\!]_i(\bar{\mathtt{R}}) \subseteq [\![\mathtt{Q}']\!]_i(\bar{\mathtt{R}})$ *holds.*

If the set Σ of constraints in the above definition is not specified (or it always evaluates to `true`), then the two problems are simply called *containment problem* and *disjunctive containment problem*. Note that the (constrained) disjunctive containment problem is more general than the (constrained) containment problem.

The first ingredient of the proof of Theorem 1 is the following proposition.

PROPOSITION 2. *The disjunctive containment problem for lefthandside CQs and righthandside* RA^+*-queries, under positive and negative containment constraints, is NP-complete.*

We will also need some basic facts about the transformation of a given RA^+-query into an equivalent union of conjunctive queries. Such a transformation, which may imply an exponential blowup, is achieved by "pushing upward" all occurrences of the union operator of the relational algebra. Formally, the transformation rules are as follows:

$$\begin{aligned}
\rho_{\{a/b\}}(\mathtt{Q}_1 \cup \mathtt{Q}_2) &\rightsquigarrow \rho_{\{a/b\}}(\mathtt{Q}_1) \cup \rho_{\{a/b\}}(\mathtt{Q}_2) \\
\sigma_c(\mathtt{Q}_1 \cup \mathtt{Q}_2) &\rightsquigarrow \sigma_c(\mathtt{Q}_1) \cup \sigma_c(\mathtt{Q}_2) \\
\pi_A(\mathtt{Q}_1 \cup \mathtt{Q}_2) &\rightsquigarrow \pi_A(\mathtt{Q}_1) \cup \pi_A(\mathtt{Q}_2) \\
(\mathtt{Q}_1 \cup \mathtt{Q}_2) \bowtie \mathtt{Q}_3 &\rightsquigarrow (\mathtt{Q}_1 \bowtie \mathtt{Q}_3) \cup (\mathtt{Q}_2 \bowtie \mathtt{Q}_3) \\
\mathtt{Q}_1 \bowtie (\mathtt{Q}_2 \cup \mathtt{Q}_3) &\rightsquigarrow (\mathtt{Q}_1 \bowtie \mathtt{Q}_2) \cup (\mathtt{Q}_1 \bowtie \mathtt{Q}_3).
\end{aligned}$$

By repeatedly applying these rules, one can transform any RA^+-query $\mathtt{Q}$ into an equivalent union of conjunctive queries of the form $\tilde{\mathtt{Q}} = \tilde{\mathtt{Q}}_1 \cup ... \cup \tilde{\mathtt{Q}}_N$, the *flattening* of $\mathtt{Q}$, where N is bounded by an exponential in the size $|\mathtt{Q}|$ of $\mathtt{Q}$ and $\tilde{\mathtt{Q}}_1, ..., \tilde{\mathtt{Q}}_N$ are conjunctive queries of size at most $|\mathtt{Q}|$. The following simple lemma shows that the problem of checking whether a given conjunctive query appears in the flattening of an RA^+-query is in NP.

LEMMA 3. *The problem of deciding, given an* RA^+*-query* $\mathtt{Q}$ *and a CQ* $\mathtt{Q}'$*, whether* $\mathtt{Q}'$ *appears as a conjunct in the flattening* $\tilde{\mathtt{Q}} = \tilde{\mathtt{Q}}_1 \cup ... \cup \tilde{\mathtt{Q}}_N$ *of* $\mathtt{Q}$ *is in NP.*

Now, it is convenient to generalize the containment relation to tuples of relations: given two tuples of relations $\bar{\mathtt{R}} = (\mathtt{R}_1, ..., \mathtt{R}_m)$ and $\bar{\mathtt{R}}' = (\mathtt{R}'_1, ..., \mathtt{R}'_m)$ of the same types, we write $\bar{\mathtt{R}} \subseteq \bar{\mathtt{R}}'$ iff $\mathtt{R}_i \subseteq \mathtt{R}'_i$ holds for all indices $1 \leq i \leq m$. Hereafter, we say that a query $\mathtt{Q}$ is *monotone* iff, for every tuples $\bar{\mathtt{R}} = (\mathtt{R}_1, ..., \mathtt{R}_m)$ and $\bar{\mathtt{R}}' = (\mathtt{R}'_1, ..., \mathtt{R}'_m)$ of relations of appropriate types, $\bar{\mathtt{R}} \subseteq \bar{\mathtt{R}}'$ implies $\mathtt{Q}(\bar{\mathtt{R}}) \subseteq \mathtt{Q}(\bar{\mathtt{R}}')$.

The last component of the proof will be the following "quantifier elimination" result for monotone queries, stating that the existence of a query satisfying certain equalities between input and output relations reduces to a boolean combination of containments between these relations.

PROPOSITION 4. *Fix* $m > 0$ *and, for all* $1 \leq i \leq m$*, let* $\bar{\mathcal{S}} \to \mathcal{T}_i$ *be an order* 1 *query type. Moreover, fix* $k > 0$ *and, for all* $1 \leq j \leq k$*, let (i)* i_j *be an index from* $\{1, ..., m\}$*, (ii)* $\bar{\mathtt{S}}_j$ *be a tuple of relations of types in* $\bar{\mathcal{S}}$*, and (iii)* $\mathtt{T}_j$ *be a relation of type* $\mathcal{T}_{i_j}$*. The following properties are equivalent:*

1. *there exist some* RA^+*-queries (or, equivalently, some UCQs)* $\mathtt{Q}_1, ..., \mathtt{Q}_m$ *such that* $\mathtt{Q}_{i_j}(\bar{\mathtt{S}}_j) = \mathtt{T}_j$ *for all* $j \in \{1, ..., k\}$*;*
2. *for every pair of indices* $j, j' \in \{1, ..., k\}$*, if* $i_j = i_{j'}$ *and* $\bar{\mathtt{S}}_j \subseteq \bar{\mathtt{S}}_{j'}$*, then* $\mathtt{T}_j \subseteq \mathtt{T}_{j'}$*.*

PROOF. The implication from 1. to 2. is trivial from the monotonicity of RA^+-queries and UCQs. The implication from 2. to 1. is proved as follows. First, we introduce, for every index $j \in \{1, ..., k\}$, a UCQ $\mathtt{Q}^{(j)}$ that, given a tuple $\bar{\mathtt{R}}$ of input relations, returns either $\mathtt{T}_j$ or the empty relation, depending on whether or not the tuple $\bar{\mathtt{S}}_j$ is contained in the tuple $\bar{\mathtt{R}}$. Note that, by construction, we have $\mathtt{Q}^{(j)}(\bar{\mathtt{S}}_j) = \mathtt{T}_j$. We then define the UCQs $\mathtt{Q}_1, ..., \mathtt{Q}_m$ as follows. For every $i^* \in \{1, ..., m\}$, $\mathtt{Q}_{i^*}$ is the union of the conjunctive queries $\mathtt{Q}^{(j)}$ over all indices j such that $i_j = i^*$. It is easy to check that property 2. implies $\mathtt{Q}_{i_j}(\bar{\mathtt{S}}_j) = \mathtt{T}_j$ for all $j \in \{1, ..., k\}$. □

Note: This result depends heavily on the presence of data constants. Characterizations of query definability with constant-free languages do exist — in the database community these date back to the work of Bancilhon [5] and Paredaens [28] (see also the recent [14], whose results bear some similarity to the proposition above). However such characterizations are more complex, and thus query definability in these other languages can not be reduced to a set of inclusion constraints.

We are now ready to prove that the higher-order containment problem is in Π_2^{P}.

PROPOSITION 5. *The problem of deciding the containment* $\Phi \subseteq_{\lambda \mathsf{RA}^+} \Phi'$*, where* $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{RA}^+]$*, is in* Π_2^{P}*.*

PROOF. We fix two order 2 terms in normal form

$$\Phi = \lambda Q_1 ... \lambda Q_m.\ \lambda R_1 ... \lambda R_n.\ \tau$$
$$\Phi' = \lambda Q_1 ... \lambda Q_m.\ \lambda R_1 ... \lambda R_n.\ \tau'$$

where each Q_i is an order 1 query variable, each R_j is a relational variable, and τ, τ' are well-typed terms of order 0 over the variables $Q_1, ..., Q_m$, $R_1, ..., R_n$ and the constants from the signature RA^+. Below, we provide a logical characterization of the non-containment relationship $\Phi \not\subseteq_{\lambda\mathsf{RA}^+} \Phi'$, that is, the existence of some queries $\mathtt{Q}_1, ..., \mathtt{Q}_m$ of the positive relational algebra and some relations $\mathtt{R}_1, ..., \mathtt{R}_n$ that witness $[\![\tau]\!](\bar{\mathtt{Q}}, \bar{\mathtt{R}}) \not\subseteq [\![\tau']\!](\bar{\mathtt{Q}}, \bar{\mathtt{R}})$.

We start by introducing new relations for the intermediate outputs produced by the subterms of τ and τ' (we explain the construction for τ only, the one for τ' is similar). We enumerate all occurrences of proper subterms of τ that are *arguments to a query variable* Q_i, for some $1 \leq i \leq m$. Let $\sigma_1, ..., \sigma_k$ be such an enumeration. Without loss of generality, we can assume that $j < j'$ holds whenever σ_j occurs inside $\sigma_{j'}$ (note that we distinguish between possible multiple occurrences of the same subterm). We then associate with each occurrence σ_j the following objects: (i) the index $i_j \in \{1, ..., m\}$ of the query variable to which σ_j is applied, (ii) two relations $\mathtt{S}_j, \mathtt{T}_j$ (of appropriate types), (iii) a term $\mathtt{P}_j$ obtained from σ_j by replacing any top-level subterm of the form $Q_{i'}(\sigma_{j'})$ by $\mathtt{T}_{j'}$. We further introduce an additional query constant $\mathtt{P}_0$, obtained from τ by replacing any top-level subterm of the form $Q_{i'}(\sigma_{j'})$ by $\mathtt{T}_{j'}$. Note that, since τ is in normal form, all its subterms are applied to query variables and query constants only. This means that each term $\mathtt{P}_j$, with $0 \leq j \leq k$, is an RA^+-query over the relations $\mathtt{R}_1, ..., \mathtt{R}_m$, $\mathtt{T}_1, ..., \mathtt{T}_k$. Analogous definitions are given for the objects i'_j, $\mathtt{S}'_j$, $\mathtt{T}'_j$, $\mathtt{P}'_j$ with respect to the occurrences of subterms in τ'.

We can now reduce the non-containment relationship $\Phi \not\subseteq \Phi'$ to the following property (for the sake of brevity, we use the shorthands $\bar{\mathtt{R}} = (\mathtt{R}_1, ..., \mathtt{R}_n)$, $\bar{\mathtt{S}} = (\mathtt{S}_0, ..., \mathtt{S}_k)$, etc.):

$$\begin{array}{l} \exists\ Q_1, ..., Q_m \\ \exists\ \bar{R}, \bar{S}, \bar{T}, \bar{S}', \bar{T}'. \quad \mathtt{P}_0(\bar{R}, \bar{T}, \bar{T}') \not\subseteq \mathtt{P}'_0(\bar{R}, \bar{T}, \bar{T}') \quad \wedge \\ \quad \bigwedge_{1 \leq j \leq k} \mathtt{P}_j(\bar{R}, \bar{T}, \bar{T}') = S_j \quad \wedge \quad \bigwedge_{1 \leq j \leq h} \mathtt{P}'_j(\bar{R}, \bar{T}, \bar{T}') = S'_j \quad \wedge \\ \quad \bigwedge_{1 \leq j \leq k} Q_{i_j}(S_j) = T_j \qquad \wedge \quad \bigwedge_{1 \leq j \leq h} Q_{i'_j}(S'_j) = T'_j. \end{array} \tag{1}$$

By exploiting Proposition 4, we can get rid of the existential quantification over $Q_1, ..., Q_m$ thus obtaining:

$$\begin{array}{l} \exists\ \bar{R},\ \bar{S},\ \bar{T},\ \bar{S}',\ \bar{T}'. \quad \mathtt{P}_0(\bar{R}, \bar{T}, \bar{T}') \not\subseteq \mathtt{P}'_0(\bar{R}, \bar{T}, \bar{T}') \quad \wedge \\ \quad \bigwedge_{1 \leq j \leq k} \mathtt{P}_j(\bar{R}, \bar{T}, \bar{T}') = S_j \qquad \wedge \quad \bigwedge_{1 \leq j \leq h} \mathtt{P}'_j(\bar{R}, \bar{T}, \bar{T}') = S'_j \quad \wedge \\ \quad \bigwedge_{\substack{1 \leq j, j' \leq k \\ i_j = i_{j'}}} S_j \subseteq S_{j'} \rightarrow T_j \subseteq T_{j'} \wedge \bigwedge_{\substack{1 \leq j, j' \leq h \\ i'_j = i'_{j'}}} S'_j \subseteq S'_{j'} \rightarrow T'_j \subseteq T'_{j'} \quad \wedge \\ \quad \bigwedge_{\substack{1 \leq j \leq k \\ 1 \leq j' \leq h \\ i_j = i'_{j'}}} S_j \subseteq S'_{j'} \rightarrow T_j \subseteq T'_{j'} \wedge \bigwedge_{\substack{1 \leq j \leq h \\ 1 \leq j' \leq k \\ i'_j = i_{j'}}} S'_j \subseteq S_{j'} \rightarrow T'_j \subseteq T_{j'}. \end{array} \tag{2}$$

It is convenient now to rename the relational variables T_j and $T'_{j'}$, where j ranges over $\{1, ..., k\}$ and j' ranges over $\{1, ..., h\}$, by new relational variables U_i, where i ranges over an appropriate set I of indices isomorphic to $\{1, ..., k\} \uplus \{1, ..., h\}$, and, similarly, replace the queries $\mathtt{P}_j(\bar{R}, \bar{T}, \bar{T}')$ and $\mathtt{P}'_j(\bar{R}, \bar{T}, \bar{T}')$ by new queries $\mathtt{O}_i(\bar{R}, \bar{U})$. Accordingly, the conditions of the form $S_j \subseteq S_{j'} \rightarrow T_j \subseteq T_{j'}$ will be replaced by equivalent conditions of the form $\mathtt{O}_i(\bar{R}, \bar{U}) \subseteq \mathtt{O}_{i'}(\bar{R}, \bar{U}) \rightarrow U_i \subseteq U_{i'}$, where the pair (i, i') is either $(0, 0)$ or an element of an appropriate subset D of $I \times I$.

Now, for every partition $\mathcal{D} = (D_+, D_-)$ of D, we denote by $\Sigma_{\mathcal{D}}$ the set of all positive constraints of the form $U_i \subseteq U_{i'}$, with $(i, i') \in D_+$, and all negative constraints of the form $U_i \not\subseteq U_{i'}$, with $(i, i') \in D_-$. Intuitively, each $\Sigma_{\mathcal{D}}$ is a maximal set of containment relationships between the various instances U_i and $U_{i'}$, for all $(i, i') \in D$. Therefore, Property (2) holds iff there exist a partition $\mathcal{D} = (D_+, D_-)$ of D such that

$$\begin{array}{l} \exists\ \bar{R}, \bar{U} \vDash \Sigma_{\mathcal{D}}. \qquad \mathtt{O}_0(\bar{R}, \bar{U}) \not\subseteq \mathtt{O}'_0(\bar{R}, \bar{U}) \quad \wedge \\ \qquad\qquad \bigwedge_{(i,i') \in D_-} \mathtt{O}_i(\bar{R}, \bar{U}) \not\subseteq \mathtt{O}_{i'}(\bar{R}, \bar{U}). \end{array} \tag{3}$$

We observe that any containment relationship of the form $\mathtt{O}_i(\bar{R}, \bar{U}) \not\subseteq \mathtt{O}'_{i'}(\bar{R}, \bar{U})$, where $\mathtt{O}_i$ is an RA^+-query, is equivalent to an existential quantification over all containment relationships of the form $\tilde{\mathtt{O}}_{i,l}(\bar{R}, \bar{U}) \not\subseteq \mathtt{O}'_{i'}(\bar{R}, \bar{U})$, where $\tilde{\mathtt{O}}_{i,l}$ is a conjunct of the flattening of $\mathtt{O}_i$. This shows that Property (3) above is *violated* (and hence $\Phi \subseteq_{\lambda\mathsf{RA}^+} \Phi'$) iff, for every partition $\mathcal{D} = (D_+, D_-)$ of D and every choice of a conjunct $\tilde{\mathtt{O}}_{0,l_0}$ from the flattening of $\mathtt{O}_0$ and for each choice of a conjunct $\tilde{\mathtt{O}}_{i,l_{i,i'}}$ from the flattening of $\mathtt{O}_i$, for each $(i, i') \in D_-$, the following instance of the constrained disjunctive containment problem is satisfied:

$$\begin{array}{l} \forall\ \bar{R}, \bar{U} \vDash \Sigma_{\mathcal{D}}. \qquad \tilde{\mathtt{O}}_{0,l_0}(\bar{R}, \bar{U}) \subseteq \mathtt{O}'_0(\bar{R}, \bar{U}) \quad \vee \\ \qquad\qquad \bigvee_{(i,i') \in D_-} \tilde{\mathtt{O}}_{i,l_{i,i'}}(\bar{R}, \bar{U}) \subseteq \mathtt{O}_{i'}(\bar{R}, \bar{U}). \end{array} \tag{4}$$

Such a characterization, together with Lemma 3 (which proves that a conjunct of the flattenings of an RA^+-query can be guessed non-deterministically in polynomial time) and Proposition 2 (which proves the NP membership for the constrained disjunctive problem with lefthandside CQs and righthandside terms RA^+-queries, under positive and negative containment constraints), shows that the problem of deciding $\Phi \subseteq_{\lambda\mathsf{RA}^+} \Phi'$ is in Π^{P}_2. □

Note that the following proposition gives immediately a Π^{P}_2-hardness result also for the higher order containment problem $\Phi \subseteq_{\lambda\mathsf{RA}^+} \Phi'$.

PROPOSITION 6. *The problem of deciding the containment* $\mathtt{Q} \subseteq \mathtt{Q}'$*, where* $\mathtt{Q}$ *is an* RA^+*-query (indeed, a* CQ_{C}*) and* $\mathtt{Q}'$ *is a CQ, is* Π^{P}_2*-hard.*

The proof of this proposition uses the same technique as the Π^{P}_2-hardness proof for the problem of deciding containment between two monotonic relational expressions, see, for instance, [32]. The above hardness result, however, strongly relies on the use of constants.

Proposition 5 and Proposition 6 together give precisely the claim of Theorem 1. Moreover, in the proof of Proposition 5, we use only a few main properties, in particular: (i) the constrained disjunctive containment for lefthandside CQs and righthandside RA^+-queries, under positive and negative containment constraints, is in NP, and (ii) the set of all possible queries that can be used to instantiate an order

1 variable is as expressive as the set of all monotone queries. Therefore, we can extend the result as follows:

COROLLARY 7. *Let* $\mathsf{RA}^{+,\neq}$ *be the signature that extends* RA^{+} *with selection operators that use equalities and inequalities between attributes, or between attributes and constants. Then, the problem of deciding the containment* $\Phi \subseteq_{\lambda\mathsf{RA}^{+,\neq}} \Phi'$*, where* $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{RA}^{+}]$*, is* Π_2^{P}*-complete.*

3.2 Adding dependencies

We now consider higher-order containment relative to *integrity constraints*. We focus on two widely-studied constraint classes, namely, functional dependencies and inclusion dependencies [1]. The containment problem for CQs under sets of functional dependencies has been deeply investigated starting from [2] and it is known to be NP-complete.

Below, given two higher-order queries $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{RA}^{+}]$ of the same type and given a set Δ of constraints (e.g., functional dependencies) over the formal arguments of Φ and Φ', we write $\Phi \subseteq_{\lambda\mathsf{RA}^{+},\Delta} \Phi'$ iff, for every input $\bar{\mathtt{Q}}, \bar{\mathtt{R}}$ that satisfies the constraints in Δ, we have $[\![\Phi]\!]_{\lambda\mathsf{RA}^{+}}(\bar{\mathtt{Q}}, \bar{\mathtt{R}}) \subseteq [\![\Phi']\!]_{\lambda\mathsf{RA}^{+}}(\bar{\mathtt{Q}}, \bar{\mathtt{R}})$.

We can extend Theorem 1 to this setting:

THEOREM 8. *The problem of deciding the containment* $\Phi \subseteq_{\lambda\mathsf{RA}^{+},\Delta} \Phi'$*, where* $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{RA}^{+}]$ *and* Δ *is a set of functional dependencies, is* Π_2^{P}*-complete.*

The proof of the complexity upper bound goes along the same lines of the proof of Proposition 5. More precisely, we first exploit Proposition 4 (which is independent of the presence of constraints on the relations) to reduce the containment problem for higher-order queries to the problem of universally guessing and deciding suitable instances of the disjunctive containment problem involving lefthandside CQs and righthandside RA^{+}-queries, under positive and negative containment constraints and the additional functional dependencies. We then argue that the latter variant of the disjunctive containment problem is in NP:

PROPOSITION 9. *The disjunctive containment problem for lefthandside CQs and righthandside* RA^{+}*-queries, under positive and negative containment constraints and functional dependencies, is NP-complete.*

The proof that Theorem 8 follows from the proposition above mimics the argument in Theorem 1.

Now, we turn towards higher-order containment in the setting of inclusion dependencies.

THEOREM 10. *The problem of deciding the containment* $\Phi \subseteq_{\lambda\mathsf{RA}^{+},\Delta} \Phi'$*, where* $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{RA}^{+}]$ *and* Δ *is a set of inclusion dependencies, is PSPACE-complete.*

PROOF. It is known that the containment problem between two CQs under a set Δ of inclusion dependencies is PSPACE-hard (see, for instance, [10]). In addition, CQs, considered as constant functionals, are special cases of higher-order queries over the signature CQ. Thus, the higher order containment problem under a set of inclusion dependencies is PSPACE-hard as well.

We now prove the PSPACE upper bound. Using the same transformation as in the proof of Theorem 1, we reduce the higher order containment problem under a set Δ of inclusion dependencies to the problem of universally guessing and deciding suitable instances of the disjunctive containment problem that have the following form:

$$\forall\, \bar{R}, \bar{U} \vDash \Sigma_{\mathcal{D}} \cup \Delta. \qquad \tilde{\mathtt{O}}_{0,l_0}(\bar{R}, \bar{U}) \subseteq \mathtt{O}'_0(\bar{R}, \bar{U}) \quad \vee \quad \bigvee_{(i,i')\in D_-} \tilde{\mathtt{O}}_{i,l_{i,i'}}(\bar{R}, \bar{U}) \subseteq \mathtt{O}_{i'}(\bar{R}, \bar{U}).$$

where $\Sigma_{\mathcal{D}}$ is a set of positive and negative containment constraints and Δ is the set of inclusion dependencies.

Now, we observe that positive containment constraints are special forms of inclusion dependencies. Thus, in order to decide the above property, it is sufficient to consider the disjunctive containment problem for lefthandside CQs and righthandside RA^{+}-queries, under negative containment constraints and inclusion dependencies. By a straightforward generalization of the proof of Proposition 2, this problem can be reduced to the containment problem for lefthandside CQs and righthandside RA^{+}-queries, under inclusion dependencies only. Finally, the latter problem can be solved in polynomial space by guessing a conjunct of the flattening of the righthandside RA^{+}-query and by deciding a classical containment problem between CQs under inclusion dependencies, which is known to be in PSPACE [19]. □

3.3 Tractable cases

We conclude this section by considering special instances of the higher-order containment problem that can be solved efficiently, namely, by a non-deterministic polynomial-time algorithm (or, even better, by a deterministic polynomial-time algorithm).

DEFINITION 4. *We define the class of* single-argument terms *as the least set that contains all terms of the form:*

- $\mathtt{Q}(R_1, ..., R_n)$*, where* $R_1, ..., R_n$ *are relational variables and* $\mathtt{Q}$ *is an* RA^{+}*-query with* n *formal arguments;*
- $\mathtt{Q}\big(Q(\tau), ..., Q(\tau)\big)$*, where* τ *is a single-argument term with at most one free query variable* Q *and* $\mathtt{Q}$ *is an* RA^{+}*-query, whose input is instantiated with as many copies of the term* $Q(\tau)$ *as the number of formal arguments of* $\mathtt{Q}$*.*

We then define single-argument higher-order queries *as the closures (by* λ*-abstraction over all free variables) of single-argument terms.*

We associate with each single-argument higher-order query Φ the (unique) sequence of RA^{+}-queries that generates the body of Φ in the grammar above, namely, the sequence $\mathtt{Q}_1, ..., \mathtt{Q}_n$ such that $\Phi = \lambda Q.\, \lambda \bar{R}.\, \mathtt{Q}_n\big(..., Q(\mathtt{Q}_{n-1}(...)), ..\big)$. We call this sequence the *generating sequence* for τ and its length the *nesting-depth* of Φ.

EXAMPLE 5. *The term* $\lambda Q.\, \lambda R.\, \rho_{a/b}\big(Q(R)\big) \bowtie \rho_{a'/b'}\big(Q(R)\big)$ *is a single-argument higher-order query, whose generating sequence consists of single* RA^{+}*-query* $\mathtt{Q}_1 = \lambda S.\, \rho_{a/b}(S) \bowtie \rho_{a'/b'}(S)$*. On the other hand, the term* $\lambda Q.\, \lambda R_1.\, \lambda R_2.\, Q(R_1) \bowtie Q(R_2)$ *is not a single-argument higher-order query, since the two formal arguments of the operator* $\bowtie$ *are instantiated with syntactically different terms.*

Hereafter, we say that a query $\mathtt{Q}$ is *non-constant* if its equivalent rule-based form has at least one variable in the head. In the special case of single-argument higher-order queries where the generating sequences consists of non-constant RA^+-queries only, we can reduce higher order containment to ordinary containment:

PROPOSITION 11. *Given two single-argument higher-order queries Φ, Φ' of the same type and with generating sequences $\mathtt{Q}_1, ..., \mathtt{Q}_m$ and $\mathtt{Q}'_1, ..., \mathtt{Q}'_n$, both consisting of non-constant RA^+-queries, we have*

$$\Phi \subseteq_{\lambda\mathsf{RA}^+} \Phi' \quad \textit{iff} \quad \begin{cases} m = n \\ \mathtt{Q}_i \subseteq \mathtt{Q}'_i \quad \textit{for all } 1 \leq i \leq m. \end{cases}$$

PROOF. As the "if" direction is trivial, we sketch the proof of the opposite direction. We assume that either $m \neq n$, or $\mathtt{Q}_i \not\subseteq \mathtt{Q}'_i$ for some $1 \leq i \leq m$ $(= n)$, and we prove that $\Phi \not\subseteq_{\lambda\mathsf{RA}^+} \Phi'$ follows. If $m \neq n$, then we introduce instances for the relational and query variables such that: (i) for all indices $1 \leq i \leq \min(m, n)$, the results of the Φ-subquery $Q(\mathtt{Q}_i(\ldots))$ and the Φ'-subquery $Q(\mathtt{Q}'_i(\ldots))$ are equivalent, and (ii) the result of Φ is not contained in the result of Φ'. In the other case, we let k be the smallest index such that $\mathtt{Q}_k \not\subseteq \mathtt{Q}'_k$. As before, we instantiate the relational and query variables with suitable values such that: (i) the results of the Φ-subquery $Q(\mathtt{Q}_i(\ldots))$ and the Φ'- subquery $Q(\mathtt{Q}'_i(\ldots))$ are equivalent for all indices $1 \leq i \leq k$, and (ii) the result of Φ is not contained in the result of Φ'. □

From Proposition 11, we immediately obtain the following result:

THEOREM 12. *The problem of deciding the containment $\Phi \subseteq_{\lambda\mathsf{RA}^+} \Phi'$, where Φ, Φ' are single-argument higher-order queries, with generating sequences consisting of non-constant UCQs, is NP-complete.*

Moreover, if we further restrict the single-argument higher-order queries in such a way that their generating sequences contain only non-constant queries in a certain tractable class, then we immediately obtain an analogous class of higher-order queries for which the containment problem turns out to be tractable (i.e., in P). For instance, consider the case of *acyclic* CQs, where evaluation becomes tractable [38]. Likewise, we have that containment of UCQs in acyclic CQs is tractable. We can then extend this to:

COROLLARY 13. *The problem of deciding the containment $\Phi \subseteq_{\lambda\mathsf{RA}^+} \Phi'$, where Φ is a single-argument higher-order query, with generating sequence consisting of non-constant UCQs, and Φ' is a single-argument higher-order query, with generating sequence consisting of non-constant acyclic CQs, is tractable.*

We can easily replace, in the above result, the acyclicity condition over order 1 queries by other conditions that guarantee tractability for ordinary conjunctive query containment (e.g., bounded treewidth, bounded hyper-treewidth, [15]).

4. HIGHER-ORDER CONTAINMENT IN OTHER BASES

We now consider the situation when we move from Positive Relational Algebra to other bases.

4.1 The general relational algebra case

In this subsection we focus on the higher-order containment problem for the case where query variables are instantiated by queries of the full relational algebra. We will still restrict the constant operators used in the higher-order queries to range over the signature RA^+, since it is well-known that the containment problem for terms built up from the full relational algebra is undecidable. In contrast to this, we show that extending the base does *not* make higher-order containment harder.

THEOREM 14. *The problem of deciding the containment $\Phi \subseteq_{\lambda\mathsf{RA}} \Phi'$, where $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{RA}^+]$, is Π_2^{P}-complete.*

The complexity lower bound is trivial from previous results. As regards the complexity upper bound, we remark here that the key ingredient, as before, is a "quantifier elimination" property, namely, the analog of Proposition 4 for queries quantified over the full Relational Algebra:

PROPOSITION 15. *Fix $m > 0$ and, for all $1 \leq i \leq m$, let $\bar{\mathcal{S}} \to \mathcal{T}_i$ be an order 1 query type. Moreover, fix $k > 0$ and, for all $1 \leq j \leq k$, let (i) i_j be an index from $\{1, ..., m\}$, (ii) $\bar{\mathtt{S}}_j$ be a tuple of relations of types in $\bar{\mathcal{S}}$, and (iii) $\mathtt{T}_j$ be a relation of type $\mathcal{T}_{i_j}$. The following properties are equivalent:*

1. *there exist some RA-queries $\mathtt{Q}_1, ..., \mathtt{Q}_m$ such that $\mathtt{Q}_{i_j}(\bar{\mathtt{S}}_j) = \mathtt{T}_j$ for all $j \in \{1, ..., k\}$;*
2. *for every pair of indices $j, j' \in \{1, ..., k\}$, if $i_j = i_{j'}$ and $\bar{\mathtt{S}}_j = \bar{\mathtt{S}}_{j'}$, then $\mathtt{T}_j = \mathtt{T}_{j'}$.*

4.2 The case of conjunctive queries

Here we show that moving to the conjunctive query base does not make the higher-order containment problem easier. Indeed, Proposition 6 gives immediately the following hardness result:

COROLLARY 16. *Let $\mathcal{I}$ be any arbitrary interpretation for the query variables (e.g., $\mathcal{I} = \lambda\mathsf{CQ}_\mathsf{C}$). The problem of deciding the containment $\Phi \subseteq_{\mathcal{I}} \Phi'$, where $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{CQ}_\mathsf{C}]$ is Π_2^{P}-hard. The lower bound holds also in the case where Φ or Φ', or both of them, contains no occurrences of query variables.*

A similar lower bound holds for the higher-order containment problem in the signature CQ:

PROPOSITION 17. *The problem of deciding the containment $\Phi \subseteq_{\lambda\mathsf{CQ}} \Phi'$, where $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{CQ}]$ and Φ' contains no occurrences of query variables, is Π_2^{P}-hard.*

The proposition is proved by using a reduction similar to the proof of Proposition 6, with the use of a query variable in Φ instead of constants.

Of course, the hardness result does not hold in the symmetric case, where the lefthandside higher-order query has no occurrences of query variables:

PROPOSITION 18. *The problem of deciding the containment $\Phi \subseteq_{\lambda\mathsf{CQ}} \Phi'$, where $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{CQ}]$ and Φ contains no occurrences of query variables, is NP-complete.*

PROOF. By monotonicity, it suffices to show that containment hold when all the query variables in Φ' return $\emptyset$. Thus, we can reduce this problem to the containment problem between two CQs, which is known to be NP-complete. □

As for the upper bounds, at the moment, we are only able to provide a result that matches with Proposition 17:

PROPOSITION 19. *The problem of deciding the containment* $\Phi \subseteq_{\lambda\mathsf{CQ}} \Phi'$*, where* $\Phi, \Phi' \in \mathsf{Terms}_2^{\downarrow}[\mathsf{CQ}]$ *and* Φ' *contains no occurrences of query variables, is in* Π_2^{P}.

The proof of the above result is based on the idea that, in order to decide the containment $\Phi \subseteq_{\lambda\mathsf{CQ}} \Phi'$, it is sufficient to consider instantiations of query variables having size bounded by a polynomial in the size of the input terms.

5. UNNORMALIZED TERMS

Our results on higher-order containment have focused on terms in normal form. We now discuss the situation for non-normalized terms. Note that the issues dealt with in the previous sections were fairly independent of the syntax of the calculus, depending rather on the range of query variables – they involve reasoning about the existence of queries having certain properties, which is our main interest. Unnormalized terms have an additional source of complexity, related to the phenomenon of *sharing* subterms during β-reductions; it is exactly the source of complexity that is eliminated in considering normalized terms.

We examine this in isolation from the prior issue, by focusing on questions about terms of order at most 1, that is, terms that evaluate to either relations or queries, rather than representing functionals. We recall that the set of relational (resp., query) closed terms of degree 1, over a signature $\mathcal{F}$, is denoted by $\mathsf{Terms}_{0,1}[\mathcal{F}]$ (resp., $\mathsf{Terms}_{1,1}[\mathcal{F}]$). All of the tight bounds we have are for unnormalized terms of degree 1.

5.1 Succinctness of unnormalized terms

We start by explaining that sharing of subterms can make unnormalized terms much more succinct than their normalized counterparts. From a standard argument in functional programming (similar results occur in the context of nested relational algebra and functional query languages, see e.g. [20]) one can see that terms that use query and relation variables are much more succinct than simple RA^+-terms. What is less well-noted, perhaps, is that the same holds for degree 1 terms with respect to "flat" unions of conjunctive queries. That is:

PROPOSITION 20. *There are terms* $\Phi_n \in \mathsf{Terms}_{1,2}[\mathsf{CQ}]$ *(i.e. using query variables but evaluating to a query) of size* $O(n)$ *where any equivalent* RA^+*-query is of size at least* 2^{2^n}*. There are such terms in* $\mathsf{Terms}_{1,1}[\mathsf{RA}^+]$ *such that any equivalent union of conjunctive queries is of size at least* 2^{2^n}.

PROOF. As for the first part, we observe that the calculus allows terms of degree 2 and size $\mathcal{O}(n)$ that check for the existence of a path of length 2^n in the directed graph represented by a given binary relation R. An example of such a term is $\varphi_n = \lambda R.\,[n](\mathtt{Q})(R)$, where $[n] = \lambda Q.\,\lambda R.\,Q^n(R)$ is a typed variant of a Church numeral and $\mathtt{Q}$ is a conjunctive query (i.e., a simple CQ-term) that maps a binary relation $\mathtt{R}$ to the composition $\mathtt{R} \circ \mathtt{R} = \{(x,z) : \exists\, y\,.\,(x,y) \in \mathtt{R},\ (y,z) \in \mathtt{R}\}$. Moreover, the degree 2 term $\Phi_n = \lambda R.\,(\varphi_2 \circ ... \circ \varphi_2)(R)$, where $\varphi_2 \circ \varphi_2$ is a shorthand for the functional composition $\lambda R.\varphi_2(\varphi_2(R))$, is equivalent (up to β-reduction) to a term of degree 2 of the form $\lambda R.\,[2^n](\mathtt{Q})(R)$. This term can check for the existence of a path of length 2^{2^n} in a given binary relation R. An Ehrenfeucht-Fraïsse game argument finally shows that any RA^+-query with less than 2^{2^n} variables cannot check this.

As for the second part, let A and B be two unary predicates and let R be a binary predicate. Let Φ_n be a query term of degree 1 that checks whether the graph represented by the binary relation R contains a path of length 2^n consisting of nodes satisfying $A \vee B$. One can easily write this with a term of size $\mathcal{O}(n)$. Now, consider a UCQ Φ'_n equivalent to Φ_n. Each disjunct D_i in Φ'_n consists of a collection of existentially quantified variables $\vec{x}$ followed by a conjunction C_i. Note that for any path π of size 2^n, there is a model R_π that has that has an isomorphic copy of that path and no other path of this size. For every such path π, let D_π be the disjunct that is satisfied in the corresponding model. Clearly, any two non-isomorphic paths π and π' have distinct corresponding disjuncts D_π and $D_{\pi'}$. This shows that any UCQ Φ'_n equivalent to Φ_n contains doubly exponentially many disjuncts. □

5.2 Expressiveness of terms of degree 1

We now show that degree 1 terms are actually familiar objects in database querying. Recall that *Datalog* queries over an input schema S consist of a collection of *intensional predicates* P and a finite set of rules of the form $H(\vec{x}) \leftarrow B(\vec{x})$, where each x_i is either a constant or a variable, the $B(\vec{x})$ are conjunctive queries over $P \cup S$, and the head predicates H are intensional predicates. A Datalog query is *non-recursive* if the dependency relation between intensional predicates is acyclic. Datalog with Stratified Negation allows the bodies $B(\vec{x})$ to contain negated predicates, but with the acyclicity criterion preserved. In the proposition below, we focus on boolean Datalog queries, in which there is a distinguished 0-ary goal predicate; the query returns true on an instance iff the goal predicate is satisfied. The following is easy to show, simply by translating between relational variables to intensional predicates:

PROPOSITION 21. *There are polynomial translations between:*

1. $\mathsf{Terms}_{1,1}[\mathsf{RA}]$ *and Nonrecursive Datalog with Stratified Negation*
2. $\mathsf{Terms}_{1,1}[\mathsf{RA}^+]$ *and Nonrecursive Datalog*
3. $\mathsf{Terms}_{1,1}[\mathsf{CQ}]$ *and Nonrecursive Datalog in which every intensional predicate occurs on the lefthandside of at most one rule.*

For brevity we avoid stating the similar characterization for $\mathsf{CQ_C}$, or the extension to the non-boolean case. Note that Nonrecursive Datalog with Stratified Negation can be translated in polynomial time (over models of size two) into first-order logic or relational algebra [4, 36]. Nonrecursive Datalog translates into positive existential first-order logic in (provably worst case) exponential time; this in turn translates into Unions of Conjunctive Queries, again in exponential time. The earlier propositions indicate that this blow-up is essential.

5.3 Complexity of terms of degree 1

We now turn to the complexity of evaluation of unnormalized terms of order 0 and degree 1 (namely, relational terms where all variables have relational type) and of containment between unnormalized terms of order 1 and degree

1 (namely, query terms defined using λ-abstraction over relational variables only).

We begin by dealing with the *evaluation problem*. Precisely, we want to decide, given a closed term Φ of relational type τ and degree 1 and given a tuple $t \in \mathcal{D}om(\tau)$, whether t belongs to the evaluation $[\![\Phi]\!]$ of Φ. The following complexity result for the evaluation problem stems form Proposition 21 and from known results in the literature.

PROPOSITION 22. *The problem of evaluating $[\![\Phi]\!]$, where $\Phi \in \mathsf{Terms}_{0,1}[\mathsf{RA}]$, is PSPACE-complete.*

Indeed, relational terms of degree 1 correspond to first-order logic formulas with "Let" definitions, i.e., built up hierarchically with equations of the form $R(\vec{x}) = \phi(\vec{x})$, where ϕ mentions only input relations and predicates defined earlier; this, in turn, is the same as Nonrecursive Datalog with Stratified Negation, which is known to be PSPACE-complete [34] (this is also credited to Immerman, perhaps because the terminology of [34] is different: see Theorem 5.3 of [13]). PSPACE-hardness is clear, since it is true for ordinary evaluation of RA-queries.

Moreover, it is also true for CQ_C terms:

PROPOSITION 23. *The problem of evaluating $[\![\Phi]\!]$, where $\Phi \in \mathsf{Terms}_{0,1}[\mathsf{CQ}_\mathsf{C}]$, is PSPACE-hard.*

A proof of the above result is by reduction from the reachability problem for synchronized products of graphs [22]: using a construction similar to the proof of Proposition 20, one can indeed write a CQ_C term of order 0 and degree 1 that checks whether two distinguished vertices are connected inside the synchronized product of a tuple of graphs (note that this property is witnessed by the existence of a path of length at most exponential in the total number of vertices of the graphs). Therefore, we can conclude that all of our evaluation problems are PSPACE-complete.

We now turn to the containment problem for terms of degree 1 and order 1. Clearly this is undecidable for RA, since even the satisfiability problem is undecidable. By Proposition 21, $\mathsf{Terms}_{1,1}[\mathsf{RA}^+]$ containment is the same as Nonrecursive Datalog containment. From unfolding the recursion, we can get an upper bound of 2EXPTIME for this problem. We do not present tight bounds for $\mathsf{Terms}_{1,1}[\mathsf{RA}^+]$ in this work — it is resolved in the subsequent paper [7]. We will focus on smaller classes of terms. We first show that containment of $\mathsf{Terms}_{1,1}[\mathsf{CQ}]$ in $\mathsf{Terms}_{1,1}[\mathsf{RA}^+]$ is in PSPACE:

PROPOSITION 24. *The problem of deciding the containment $\Phi \subseteq \Phi'$, where $\Phi \in \mathsf{Terms}_{1,1}[\mathsf{CQ}]$ and $\Phi' \in \mathsf{Terms}_{1,1}[\mathsf{RA}^+]$, is in PSPACE.*

PROOF. The intuition behind the proof of the proposition is that we can explore the unfolding of Φ in PSPACE. We make this precise by giving canonical names to variables in the unfolding.

Assume that a query $\mathtt{Q}$ is given as a set of rules $Ru_1 \dots Ru_k$ with Ru_i of the form $H_i(\vec{x}) \leftarrow \phi_i(\vec{x})$, where ϕ_i is a CQ mentioning only relations $H_j : j < i$. By a standard transformation [16] we can assume that each ϕ_i has only two occurrences of relation symbols in it. Let $[\mathtt{Q}]$ be the unfolding of $\mathtt{Q}$ as a UCQ, obtained by recursively replacing an occurrence of $H_i(\vec{x})$ with $\phi_i(\vec{x})$. A partial unfolding is any intermediate formula resulting from this process. A *name* is a sequence of pairs (i, j) with $i \leq k, j \in \{1, 2\}$ of length at most k. We associate every atom and every variable in a partial unfolding of $[\mathtt{Q}]$ with a name as follows: in the original $\mathtt{Q}$, every atom is associated with the empty name. If in partial unfolding η we replace the jth occurrence O of $H_i(\vec{x})$ in η with $\phi_i(\vec{x})$ to get η', then we associate every atom and also every variable that was introduced in η' with $name(O), (i, j)$. Note that every name is thus associated with at most one relation symbol and many variables. It is easy to show that one can check properties of names in PSPACE.

Our algorithm will now mimic the standard PSPACE algorithm for evaluating a Nonrecursive Datalog query $\mathtt{P}$ on an explicitly given database, but instead of guessing elements of the database, it guesses a $\mathtt{Q}$-name. □

Since containment is harder than evaluation, we have that the containment problem of $\mathsf{Terms}_{1,1}[\mathsf{CQ}]$ in $\mathsf{Terms}_{1,1}[\mathsf{RA}^+]$ is PSPACE-complete. More specifically, from the results on the evaluation problem, we can say that the problem is hard even when the lefthandside terms are as restricted as possible and the righthandside terms do not use unions:

COROLLARY 25. *The problem of deciding the containment $\Phi \subseteq \Phi'$, where Φ is a conjunctive query and $\Phi' \in \mathsf{Terms}_{1,1}[\mathsf{CQ}_\mathsf{C}]$, is PSPACE-hard.*

However, if we restrict the righthandside terms of the containment problem, we do get a better bound for $\mathsf{Terms}_{1,1}[\mathsf{CQ}]$. The argument also uses the idea of compact names, as in Proposition 24:

THEOREM 26. *The problem of deciding the containment $\Phi \subseteq \Phi'$, where $\Phi \in \mathsf{Terms}_{1,1}[\mathsf{CQ}]$ and Φ' is a conjunctive query, is NP-complete.*

5.4 Complexity of terms of degree 2

So far we have focused on the complexity of the evaluation and containment problems for either normalized terms of order at most 2 or unnormalized terms of degree 1. By combining these results with normalization bounds for the simply-typed λ-calculus we obtain upper bounds for analogous problems for our most general language: unnormalized terms of degree 2.

β-reduction can reduce any term of degree 2 to a term of degree 1 with at most an exponential blow-up (finer bounds can be given in terms of the nesting of applications in the term, see [6]). Thus Proposition 24 immediately yields an EXPSPACE upper bound for the evaluation and the containment problems for terms in $\mathsf{Terms}_{1,2}[\mathsf{CQ}]$.

Similarly, reduction can be applied to get rid of unreduced abstractions of degrees one and two, in doubly-exponential time. Thus using Theorem 1, we obtain that the containment problem for terms in $\mathsf{Terms}_{2,2}[\mathsf{RA}^+]$ is in 2EXPSPACE.

6. RELATED WORK

This paper is related to several lines of research in the database community – both on database and programming language integration and on querying metadata. We highlight differences below.

λ-calculus and database query languages. One inspiration for our work comes from functional databases [18, 9, 27] which aim toward unification of database query languages with functional programming. Kannelakis and his

collaborators [18, 17] investigated embeddings of relational query languages into typed λ-calculi. The goal is to code the operational semantics of relational query languages in the standard reduction operations of the host calculus. [18, 17] give polynomial time encodings of standard languages, including query languages with recursion mechanisms, within variants of the λ-calculus. In contrast, in our work we do not *reduce* querying to β-reduction, we simply combine querying and reduction: relational operators are treated as fixed constants, with their usual semantics, and we deal with database instances as constants, not via encodings. Our queries have low data complexity (e.g. within AC^0), and thus can not simulate list iteration and other recursion mechanisms.

Languages such as Machiavelli [27] and Kleisli [37] embed database operations in a general-purpose functional language (e.g. ML in both cases above). The type system of the host language is extended with type constructors for various relational and object-oriented database features: e.g. records, variant records, sets. Higher-order functions can be formed and applied using the constructs of the host language; in particular, the type system can constrain the domain and range of a function on database instances, but the computational power of such functions is limited only by the host language. In contrast, our languages restrict function variables to range over query languages with clearly limited expressive power.

The Monad Algebra of [33] is presented as a λ-calculus over a type system capturing nested relational structures. Rather than embed into a general-purpose calculus, they allow functions to be built up via a collection of nested relational operators. Koch has shown that these languages are equivalent (modulo coding issues) to the functional XML query language XQuery [20]. The expressive power of queries that can arise in a nested relational language is thus bounded: for example, the well-known conservativity theorem of Paredaens and Van Gucht [29] implies that the expressive power of such a language on relational data is no more than that of relational calculus. The positive variant of Monad Algebra, defined also in [20], is analogous to our languages. However the presence of nesting operators gives nested relational languages the ability to build new values from the database – an ability our query language does not have – and this has implications for complexity. Our degree one terms are much weaker than Nested Relational Algebra (NRA) expressions; they correspond merely to first-order logic with let bindings, which can be converted tractably to ordinary relational algebra expressions (on models of size > 1 [4]). Koch has shown (modulo complexity-theoretic assumptions) [20] that this can not be done for nested relational algebra terms. On the other hand, our degree two terms are not efficiently translatable to NRA terms: they can check for the existence of a doubly-exponential sized path in a graph. In contrast, it follows from [8] that positive Monad Algebra terms can be converted in exponential time to flat existential first-order queries. Using games one can derive that such term cannot check for doubly-exponential sized paths. [20] has shown that the evaluation problem is NEXPTIME-hard even for the positive fragment of Monad Algebra.

The equivalence problems we deal with in Sections 3 and 4 have (to our knowledge) no natural analog in the existing functional query literature. For example, in the Monad Algebra of [33] all variables range over database instances – query variables and λ-abstraction over queries are not supported.

Containment and equivalence for extensions of the relational model. Query equivalence and containment has been studied extensively for many relational query classes: e.g. conjunctive queries and union of conjunctive queries, starting with [12]. There is also work for NRA and other Complex object models. [25] investigates containment and equivalence in a complex object analog of conjunctive queries, referred to as "Conjunctive Idealized Algol". There are several possible notions of containment and equivalence in this setting: [25] define a notion of simulation that corresponds roughly to our notion of higher-order containment. However, our data model does not include nesting explicitly, and we do not know of any coding of nested relations as functions that allows one to reduce Conjunctive Idealized Algol equivalence to Higher-Order query equivalence.

Meta-data and higher-order querying. Several researchers have looked at the issue of uniformly handling data and metadata within a query language – particularly see [23, 26, 30, 31]. The emphasis in most of these works is on queries that include relation names and column info in the input output, in manipulating relational queries. An exception is the work of Neven et. al. in [26], which gives a language that can manipulate tables containing both queries and data. The language of [26] is much more powerful than ours, and extends standard query languages in an intuitive way. But they do not satisfy either of our two design goals, since they are relationally complete and allow one to access the syntactic structure of queries.

Query specification. Recently there has been considerable interest in query specification formalisms [24, 35, 11]. The motivation is to describe the conjunctive queries that are supported by a particular external source. In the prior formalisms the query is specified by describing its syntax; for example, [24, 35] use a variant of Datalog to describe the structure of a family of parameterized queries. In contrast, our formalisms do not allow access to the syntax of a query.

7. CONCLUSIONS

We have defined a family of languages which can define ordinary queries and also query functionals, generalizing traditional CQs and Unions of CQs. Our languages have two advantages: the output of a query transformation depends only on the semantics of the input queries, and many basic analysis problems are decidable.

In particular, we have tight bounds on the complexity of equivalence for normal-form terms when the base is positive relational algebra. For general terms over this base, we get upper bounds by combining standard λ-calculus normalization with results on special cases of Nonrecursive Datalog containment. In this paper we have not given a complete picture of the complexity for terms of order 1 and degree 1 – that is, for Nonrecursive Datalog containment. However, subsequently this problem has been resolved [7].

The open problems are manifold. In particular, we do not have tight bounds for equivalence of unrestricted terms, even those that simply transform data to data. Furthermore, there are two natural bases where we do not have upper bounds even for containment of normal-form terms of order 2: conjunctive queries, and unions of conjunctive queries without data constants. Finally, we have not investigated

generalizations of this formalism to arbitrary orders – we plan to tackle this in future work.

Acknowledgements. We are very grateful to the anonymous referees of PODS for helpful comments and corrections. We thank TJ Green for suggestions and references that improved the camera-ready. Benedikt and Puppis are supported in part by EPSRC EP/G004021/1 (the Engineering and Physical Sciences Research Council, UK).

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient optimization of a class of relational expressions. *ACM TODS*, 4(4), 1979.

[3] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree Pattern Relaxation. In *EDBT*, 2002.

[4] J. Avigad. Eliminating Definitions and Skolem Functions in First-order Logic. *ACM TOCL*, 4(3):402–415, 2003.

[5] F. Bancilhon. On the completeness of query languages for relational data bases. In *MFCS*, 1978.

[6] A. Beckmann. Exact Bounds for Lengths of Reductions in Typed λ-Calculus. *J. Symb. Log.*, 66(3):1277–1285, 2001.

[7] M. Benedikt and G. Gottlob. The Impact of Views on Containment, 2010. Manuscript in preparation.

[8] M. Benedikt and C. Koch. From XQuery to Relational Logics. *ACM TODS*, 2009.

[9] P. Buneman and R. Frankel. FQL: a Functional Query Language. In *SIGMOD*, 1979.

[10] M. Casanova, R. Fagin, and C. Papadimitriou. Inclusion Dependencies and Their Interaction with Functional Dependencies. *JCSS*, 28(1):29–59, 1984.

[11] B. Cautis, A. Deutsch, and N. Onose. Querying Data Sources that Export Infinite sets of Views. In *ICDT*, 2009.

[12] A. Chandra and P. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*, 1977.

[13] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Comp. Surv.*, 33(3):374–425, 2001.

[14] G. H. L. Fletcher, M. Gyssens, J. Paredaens, and D. V. Gucht. On the expressive power of the relational algebra on finite sets of relation pairs. *IEEE Trans. Knowl. Data Eng.*, 21(6):939–942, 2009.

[15] G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. *JCSS*, 64(3):579–627, 2002.

[16] G. Gottlob and C. Papadimitriou. On the Complexity of Single-rule Datalog Queries. *Inf. Comput.*, 183(1), 2003.

[17] G. Hillebrand and P. Kanellakis. Functional Database Query Languages as Typed Lambda Calculi of Fixed Order. In *PODS*, 1994.

[18] G. Hillebrand, P. Kanellakis, and H. Mairson. Database Query Languages Embedded in the Typed Lambda Calculus. In *LICS*, 1993.

[19] D. S. Johnson and A. C. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *JCSS*, 28(1), 1984.

[20] C. Koch. On the Complexity of Nonrecursive XQuery and Functional Query Languages on Complex Values. *ACM TODS*, 31(4):1215–1256, 2006.

[21] N. Koudas, C. Li, A. Tung, and R. Vernica. Relaxing Join and Selection Queries. In *VLDB*, 2006.

[22] D. Kozen. Lower Bounds for Natural Proof Systems. In *FOCS*, 1977.

[23] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *DOOD*, 1993.

[24] A. Levy, A. Rajaraman, and J. Ullman. Answering Queries using Limited External Query Processors. In *PODS*, 1996.

[25] A. Levy and D. Suciu. Deciding Containment for Queries with Complex Objects. In *PODS*, 1997.

[26] F. Neven, D. Van Gucht, J. Van den Bussche, and G. Vossen. Typed query languages for databases containing queries. In *PODS*, 1998.

[27] A. Ohori, P. Buneman, and V. Breazu-Tannen. Database programming in Machiavelli—a polymorphic language with static type inference. In *SIGMOD*, 1989.

[28] J. Paredaens. On the expressive power of the relational algebra. *Inf. Process. Lett.*, 7(2):107–111, 1978.

[29] J. Paredaens and D. Van Gucht. Converting Nested Algebra Expressions into Flat Algebra Expressions. *ACM TODS*, 17(1):65–93, 1992.

[30] K. A. Ross. Relations with relation names as arguments: algebra and calculus. In *PODS*, 1992.

[31] K. A. Ross. On negation in HiLog. In *J. Log. Program.*, 1994.

[32] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

[33] V. Tannen, P. Buneman, and L. Wong. Naturally Embedded Query Languages. In *ICDT*, 1992.

[34] M. Y. Vardi. The Complexity of Relational Query Languages. In *STOC*, 1982.

[35] V. Vassalos and Y. Papakonstantinou. Expressive Capabilities Description Languages and Query Rewriting Algorithms. *The Journal of Logic Programming*, 43(1):75 – 122, 2000.

[36] S. Vorobyov and A. Voronkov. Complexity of nonrecursive logic programs with complex values. Technical Report MPI-I-97-2-010, Max-Planck Institut fiir Informatik, Saarbriicken, November 1997.

[37] L. Wong. Kleisli, a functional query system. *J. Funct. Program.*, 10(1):19–56, 2000.

[38] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, 1981.